**PRU UART Firmware**

Applies to Product Release: 01.00.00.00

Publication Date: January 31, 2019

**Document License**

This work is licensed under the Creative Commons Attribution-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

**Contributors to this document**

Copyright (C) 2019 Texas Instruments Incorporated - http://www.ti.com/



Texas Instruments, Incorporated  
12500 TI Blvd  
Dallas, TX 75243 USA

Texas Instruments, Incorporated  
20450 Century Boulevard  
Germantown, MD 20874 USA

**This document is intended for users interested the PRU UART firmware design. It discusses details of the UART firmware design and implementation, and includes information concerning the memory maps, structures, state machines, and software flow of the firmware.**

**Note: Those only concerned with using the PRU UART firmware may not need read this document.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | | |
| **Version** | **Date** | **Description of changes** | **Author(s)** |
| 0.1 | 07-09-18 | Initial Draft | Jason Reeder |
| 0.2 | 07-12-18 | Add interrupt generation to the end of the RX flow | Jason Reeder |
| 0.5 | 09-11-18 | Added the PIN\_CFG and INT\_CFG registers to pull the pin numbers and event numbers from DRAM. Added a TX buffer empty interrupt. Increased the cycles per tick to allow for the state to be saved and restored from DRAM. | Jason Reeder |
| 0.6 | 01-24-19 | Updated definitions of UART FW registers for pin & interrupt configuration.  Updated firmware TX state machine for TX buffer empty interrupt.  Added TX & RX interrupt enable & disable.  Added firmware PRU register usage map.  Added UART FW global control/ status register.  Added RESET states to firmware TX & RX state machines.  Added firmware resource requirements.  Added firmware testing. | Frank Livingston |
| 1.0 | 01-31-19 | Fixed typos in Table 22.  Added P8.2 for FTDI Cable Pin1 ground connection in Figure 5.  Added firmware version registers.  Added details concerning R/W type of firmware registers.  Added section on Unit Test Build. | Frank Livingston |

**TABLE OF CONTENTS**

LIST OF FIGURES 5

LIST OF TABLES 6

1 Introduction 7

2 Feature Set 7

3 Design Description 8

3.1 Firmware Registers 8

3.1.1 Register Memory Map 8

3.1.2 Register Description 9

3.1.2.1 UART\_GCFG 10

3.1.2.2 UART<N>\_CTRL 10

3.1.2.3 UART<N>\_CFG 10

3.1.2.4 UART<N>\_RD\_WR\_PTRS 11

3.1.2.5 UART<N>\_TX\_PIN\_INT\_CFG 12

3.1.2.6 UART<N>\_RX\_PIN\_INT\_CFG 12

3.2 Design Theory 12

3.2.1 Top Level Flow 13

3.2.2 Wait for Next Tick 14

3.2.3 Push Transmit Pin Updates & Grab Receive Pin States 15

3.2.4 State Machines 15

3.2.4.1 Transmit State Machine 15

3.2.4.2 Receive State Machine 18

3.2.5 Circular Buffers 21

3.2.6 Binary Configuration Parsing and State Selection 21

3.3 PRU Resource Usage 22

3.3.1 PRU Registers 22

3.3.2 PRU Memory 29

3.3.3 PRU Cycles 29

3.3.3.1 Worst-Case Cycle Counts 29

3.3.3.2 Transmit Cycle Counts 30

3.3.3.3 Receive Cycle Counts 31

3.3.3.4 Wait Tick Cycle Counts 31

3.4 Firmware Source Code 32

3.4.1 Firmware Macros Description 32

3.4.2 Firmware Sources Description 32

4 RTOS Driver Support 33

4.1 External APIs 33

4.2 Internal Files & Functions 34

5 Test Plan 36

5.1 EVM Support 36

5.1.1 bbbAM335x 36

5.2 Test Setup 36

5.2.1 bbbAM335x 36

5.3 Unit Test 37

LIST OF FIGURES

[Figure 1. UART Firmware Design Layers 8](#_Toc536130748)

[Figure 2. PRU UART Firmware Top-Level Program Flow 14](#_Toc536130749)

[Figure 3. UART TX State Machine 17](#_Toc536130750)

[Figure 4. UART RX State Machine 21](#_Toc536130751)

[Figure 5. PRU Firmware Unit Test Setup 37](#_Toc536130752)

LIST OF TABLES

[Table 1. Features Comparison Hard vs. Soft IP 7](#_Toc536130753)

[Table 2. UART Firmware Register Memory Map 9](#_Toc536130754)

[Table 3. UART Firmware Register Descriptions 9](#_Toc536130755)

[Table 4. UART\_CFG Firmware Register Description 10](#_Toc536130756)

[Table 5. UART<N>\_CTRL Firmware Register Description 10](#_Toc536130757)

[Table 6. UART<N>\_CFG Firmware Register Description 11](#_Toc536130758)

[Table 7. UART<N>\_RD\_WR\_PTRS Firmware Register Description 12](#_Toc536130759)

[Table 8. UART<N>\_TX\_PIN\_INT\_CFG Firmware Register Description 12](#_Toc536130760)

[Table 9. UART<N>\_RX\_PIN\_INT\_CFG Firmware Register Description 12](#_Toc536130761)

[Table 10. PRU UART Firmware Register Map 23](#_Toc536130762)

[Table 11. PRU UART Firmware Register Usage 29](#_Toc536130763)

[Table 12. PRU UART Firmware IMEM Requirements 29](#_Toc536130764)

[Table 13. PRU UART Firmware DMEM Requirements 29](#_Toc536130765)

[Table 14. Firmware Cycle Counts Equation Terms 30](#_Toc536130766)

[Table 15. Worst-Case UART Transmit State Cycles 30](#_Toc536130767)

[Table 16. Worst-Case UART Receive State Cycles 31](#_Toc536130768)

[Table 17. Firmware Macros 32](#_Toc536130769)

[Table 18. Firmware Source Files 32](#_Toc536130770)

[Table 19. UART LLD API Functions 33](#_Toc536130771)

[Table 20. UART LLD API Function Details 34](#_Toc536130772)

[Table 21. UART LLD Software IP Internal Functions 35](#_Toc536130773)

[Table 22. bbbAM335x UART Instances 36](#_Toc536130774)

# Introduction

The UART (Universal Asynchronous Receiver-Transmitter) is an asynchronous serial bus that provides a communication link between integrated circuits (ICs). Most of TI’s SoCs have their own dedicated hardware UART IP. In case of need of more UART instance than supported via hardware, firmware based UART soft IP can be used.

Firmware is designed to run on PRU cores. PRU cores have their own GPI/GPO pins which can be toggled at specific time intervals in order to implement the UART protocol. PRU programming is done using assembly instructions. PRU cores also have their internal PRU cycle counter to meet timing requirements. Host cores in the SOC configure the PRU for receiving and sending data on the PRU UART interfaces.

# Feature Set

The motivation for this firmware is to use the PRU to extend the number of available UARTs on TI SOCs incorporating PRU-ICSS in case more UARTs are desired than are provided by the SOC hardware. Given this motivation, the goal is to support the PRU UART firmware on AM3x, AM4x, AM5x and K2Gx devices.

The following UART specifications are supported by the PRU UART firmware:

* Up to 3 UARTs are supported per PRU with the following configuration options
  + Baud rate: common baud rates up to 115200
  + Bits per transfer: 5-9 bit characters
  + Number of stop bits: 1, 1.5, 2
  + Parity: even, odd, none
  + Flow control: HW, none (SW flow control currently unsupported)
  + Full-duplex communication
* Each UART can simultaneously support different configurations
  + UART0 can have a different baud rate, number of stop bits, etc. when compared to UART1 or UART2 in the same PRU
* Firmware is self-contained in a single PRU
  + No ICSS resources should are used (IEP Timer, Scratchpad, etc.)
  + Other PRUs in the ICSS are available for other functions if 3 or less UARTs are needed

Table 1 shows a comparison of the features supported by the UART firmware with those available on hardware IP.

| **UART Supported Features** | **Hardware IP** | **Firmware** |
| --- | --- | --- |
| Number of hardware instances | SoC dependent | 3 (per PRU) |
| Baud rates | SoC dependent (up to 12Mbps) | All common rates between 300‑115200 |
| Bits per character | 5-8 bits | 5‑9 bits |
| Number of stop bits | 1, 1.5, 2 | 1, 1.5, 2 |
| Parity types | Even, Odd, Mark, Space, none | Even, Odd, none |
| Flow control types | HW, SW, none | HW, none |

Table 1. Features Comparison Hard vs. Soft IP

# Design Description



Figure 1. UART Firmware Design Layers

## Firmware Registers

### Register Memory Map

The register memory map for UART firmware executing on a particular PRU is located in the data memory associated with the PRU, i.e. the map is located in DRAM0 for PRU0 and DRAM1 for PRU1.

| **Memory offset** | **Size (Bytes)** | **Register name** |
| --- | --- | --- |
| 0x00000000 | 4 | Firmware Magic Number |
| 0x00000004 | 4 | Firmware Type |
| 0x00000008 | 4 | Firmware Version |
| 0x0000000C | 4 | Firmware Feature |
| 0x00000010 | 4 | Firmware Extended Feature |
| 0x00000014 | 4 | UART0\_CTRL |
| 0x00000018 | 4 | UART0\_CFG |
| 0x0000001C | 4 | UART0\_RD\_WR\_PTRS |
| 0x00000020 | 4 | UART0\_TX\_PIN\_INT\_CFG |
| 0x00000024 | 4 | UART0\_RX\_PIN\_INT\_CFG |
| 0x00000028 | 4 | UART1\_CTRL |
| 0x0000002C | 4 | UART1\_CFG |
| 0x00000030 | 4 | UART1\_RD\_WR\_PTRS |
| 0x00000034 | 4 | UART1\_TX\_PIN\_INT\_CFG |
| 0x00000038 | 4 | UART1\_RX\_PIN\_INT\_CFG |
| 0x0000003C | 4 | UART2\_CTRL |
| 0x00000040 | 4 | UART2\_CFG |
| 0x00000044 | 4 | UART2\_RD\_WR\_PTRS |
| 0x00000048 | 4 | UART2\_TX\_PIN\_INT\_CFG |
| 0x0000004C | 4 | UART2\_RX\_PIN\_INT\_CFG |
| 0x00000050 | 4 | UART\_GCFG |
| 0x00000054 | 172 | Reserved |
| 0x00000100 | 256 | UART0\_TX\_BUFFER |
| 0x00000200 | 256 | UART0\_RX\_BUFFER |
| 0x00000300 | 256 | UART1\_TX\_BUFFER |
| 0x00000400 | 256 | UART1\_RX\_BUFFER |
| 0x00000500 | 256 | UART2\_TX\_BUFFER |
| 0x00000600 | 256 | UART2\_RX\_BUFFER |
| 0x00000700 | Remaining | Reserved |

Table 2. UART Firmware Register Memory Map

### Register Description

Following are descriptions of the firmware registers. <N> indicates the instance number of UART firmware.

|  |  |
| --- | --- |
| **Register name** | **Description** |
| **Firmware Magic Number** | **Contains magic number for PRU UART firmware.** |
| **Firmware Type** | **Contains firmware type information.** |
| **Firmware Version** | **Contains firmware version information.** |
| **Firmware Feature** | **Contains firmware feature information.** |
| **Firmware Extended Feature** | **Contains firmware extended feature information.** |
| **UART\_GCFG** | **Contains UART global configuration & status.** |
| **UART<N>\_CTRL** | **Contains UART instance control & status.** |
| **UART<N>\_CFG** | **Contains all of the configuration information for the UART protocol (baud rate, bits per character, number of stop bits, etc.).** |
| **UART<N>\_RD\_WR\_PTRS** | **Contains the current read and write pointers for both the transmit and receive buffers for the UART instance.** |
| **UART<N>\_TX\_PIN\_INT\_CFG** | **Contains the GPI/O pin number to be used for each UART RX function (TX, CTS). Allows the Host application to provide the pin numbers and make the firmware more flexible.**  **Contains the PRU-ICSS Event Number to be used for the Tx Buffer empty event.** |
| **UART<N>\_RX\_PIN\_INT\_CFG** | **Contains the GPI/O pin number to be used for each UART TX function (RX, RTS). Allows the Host application to provide the pin numbers and make the firmware more flexible.**  **Contains the PRU-ICSS Event Number to be used for the Rx character received event.** |
| **UART<N>\_TX\_BUFFER** | **256 byte buffer containing the characters to be transmitted by the UART instance. In order to support 9 bit characters, each buffer slot is 16 bits. Can hold up to 127 characters (2 bytes per character and one character slot sacrificed for ease of pointer use).** |
| **UART<N>\_RX\_BUFFER** | **256 byte buffer containing the received characters along with any parity or framing errors detected. In order to support 9 bit characters, each buffer slot is 16 bits. Can hold up to 127 characters (2 bytes per character and one character slot sacrificed for ease of pointer use).** |

Table 3. UART Firmware Register Descriptions

#### Firmware Magic Number

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-24** | **Magic Number Byte 3** | **R-54h** | **ASCII 54h – ‘T’** |
| **23-16** | **Magic Number Byte 2** | **R-52h** | **ASCII 52h – ‘R’** |
| **15-8** | **Magic Number Byte 1** | **R-41h** | **ASCII 41h – ‘A’** |
| **7-0** | **Magic Number Byte 0** | **R-55h** | **ASCII 55h – ‘U’** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware load

Table 4. Firmware Magic Number Register Description

#### Firmware Type

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-24** | **Firmware ICSS version** | **R-00h** | **00h – ICSS Revision 1** |
| **23-8** | **Firmware Protocol Type, Control Class** | **R-02h** | **02h – UART protocol** |
| **7-0** | **Firmware Protocol Type, UART Protocol Version** | **R-01h** | **01h – UART version (arbitrary, no UART specification)** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware load

Table 5. Firmware Type Register Description

#### Firmware Version

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31** | **Firmware Release or Internal Version** | **R-00h** | **00h – Release version**  **01h – Internal version** |
| **30-24** | **Firmware Version Major** | **R-MMh** | **MMh – major version number** |
| **23-8** | **Firmware Version Minor** | **R-NNh** | **NNh – minor version number** |
| **7-0** | **Firmware Version Build** | **R-BBh** | **BBh – build version number** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware load

Table 6. Firmware Version Register Description

#### Firmware Feature

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **0-31** | **Bit-map for features supported in firmware** | **R-0h** | **Reserved for future use** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware load

Table 7. Firmware Feature Register Description

#### Firmware Extended Feature

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **0-31** | **Offset to extended feature structure for future use** | **R-0h** | **Reserved for future use** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware load

Table 8. Firmware Extended Feature Register Description

#### UART\_GCFG

Following are detailed descriptions of UART\_GCFG bit fields and bit field values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-2** | **RESERVED** | **-** |  |
| **1** | **Firmware initialized flag** | **R-0h** | **Firmware sets field to 0 upon execution. Firmware sets field to 1 upon successful intialization.**  **0h – Firmware not initialized**  **1h – Firmware initialized** |
| **0** | **PRU ID** | **R/W-Xh** | **Host must set field before the firmware is executed.**  **0h – PRU ID 0**  **1h – PRU ID 1** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware execution

Table 9. UART\_CFG Firmware Register Description

#### UART<N>\_CTRL

Following are detailed descriptions of UART<N>\_CTRL bit fields and bit field values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-10** | **RESERVED** | **-** | **-** |
| **9** | **Rx Enable Acknowledge** | **R-0h** | **Firmware sets field to acknowledge UART Rx enable (Host sets Enable to ‘1’) or disable (Host sets Enable field to ‘0’).**  **0h – Rx UART disable acknowledged**  **1h – Rx UART enable acknowledged** |
| **8** | **Tx Enable Acknowledge** | **R-0h** | **Firmware sets field to acknowledge UART Rx enable (Host sets Enable to ‘1’) or disable (Host sets Enable field to ‘0’).**  **0h – Tx UART disable acknowledged**  **1h – Tx UART enable acknowledged** |
| **7-1** | **RESERVED** | **-** | **-** |
| **0** | **Enable** | **R/W-0h** | **Host must set field to enable/disable UART instance.**  **0h – Disable UART**  **1h – Enable UART** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware execution

Table 10. UART<N>\_CTRL Firmware Register Description

#### UART<N>\_CFG

Following are detailed descriptions of UART<N>\_CFG bit fields and bit field values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-24** | **RESERVED** | **-** |  |
| **23-16** | **HW Flow Control Threshold** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **When HW flow control is enabled, the current RX buffer fill level is compared to this value to determine whether or not to assert the RTS pin and request the other side to stop sending data.** |
| **15-14** | **RESERVED** |  |  |
| **13** | **Flow Control Type** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – HW Flow Control**  **1h – SW Flow Control (not currently supported)** |
| **12** | **Flow Control Enable** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – No Flow Control**  **1h – Flow Control enabled** |
| **11** | **RESERVED** |  |  |
| **10** | **Parity Type** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – Even parity**  **1h – Odd parity** |
| **9** | **Parity Enable** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – Parity disabled**  **1h – Parity enable** |
| **8-7** | **Stop bit size** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – 1 character width**  **1h – 1.5 character width**  **2h – 2 character width**  **3h – RESERVED** |
| **6-4** | **Number of bits per character** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – 5 bits per character**  **1h – 6 bits per character**  **2h – 7 bits per character**  **3h – 8 bits per character**  **4h – 9 bits per character**  **7-5h - RESERVED** |
| **3-0** | **Baud rate** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – 300**  **1h – 600**  **2h – 1200**  **3h – 2400**  **4h – 4800**  **5h – 9600**  **6h – 14400**  **7h – 19200**  **8h – 28800**  **9h – 38400**  **Ah – 57600**  **Bh – 115200**  **F-Ch - RESERVED** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware execution

Table 11. UART<N>\_CFG Firmware Register Description

#### UART<N>\_RD\_WR\_PTRS

Following are detailed descriptions of UART<N>\_RD\_WR\_PTRS bit fields and bit field values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-24** | **RX Buffer Write Pointer** | **R/W-Xh** | **Firmware resets field after UART instance is enabled.**  **This value is an offset into the UART<N> RX buffer. The PRU uses this offset to determine where to write the next character into the RX buffer.** |
| **23-16** | **RX Buffer Read Pointer** | **R/W-Xh** | **Firmware resets field after UART instance is enabled.**  **This value is an offset into the UART<N> RX buffer for the next character that the host processor needs to consume. The host processor reads from this offset to get the character, consumes the character, and then updates the pointer.** |
| **15-8** | **TX Write Pointer** | **R/W-Xh** | **Firmware resets field after UART instance is enabled.**  **This value is an offset into the UART<N> TX buffer. The host processor uses this offset to determine where to write the next character into the TX buffer.** |
| **7-0** | **TX Read Pointer** | **R/W-Xh** | **Firmware resets field after UART instance is enabled.**  **This value is an offset into the UART<N> TX buffer for the next character that the PRU needs to send. The PRU reads from this offset to get the character, sends the character, and then updates the pointer.** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware execution

Table 12. UART<N>\_RD\_WR\_PTRS Firmware Register Description

#### UART<N>\_TX\_PIN\_INT\_CFG

Following are detailed descriptions of UART<N>\_TX\_PIN\_INT\_CFG bit fields and bit field values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-24** | **CTS pin** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **PRU pin number to be used for UART CTS** |
| **23-16** | **TX pin** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **PRU pin number to be used for UART TX** |
| **15-8** | **TX buffer empty interrupt control** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – Interrupt disabled**  **1h – Interrupt enabled** |
| **7-0** | **TX event number** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **PRU-ICSS Event Number to be used for the Tx Buffer empty event** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware execution

Table 13. UART<N>\_TX\_PIN\_INT\_CFG Firmware Register Description

#### UART<N>\_RX\_PIN\_INT\_CFG

Following are detailed descriptions of UART<N>\_TX\_PIN\_INT\_CFG bit fields and bit field values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Field** | **Type** | **Description** |
| **31-24** | **RTS pin** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **PRU pin number to be used for UART RTS** |
| **23-16** | **RX pin** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **PRU pin number to be used for UART RX** |
| **15-8** | **RX character received interrupt control** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **0h – Interrupt disabled**  **1h – Interrupt enabled** |
| **7-0** | **RX event number** | **R/W-Xh** | **Host must set field before UART instance is enabled.**  **PRU-ICSS Event Number to be used for Rx character received event** |

LEGEND: R/W = Read/Write; R = Read Only; -*n* = value after firmware execution

Table 14. UART<N>\_RX\_PIN\_INT\_CFG Firmware Register Description

## Design Theory

Due to the asynchronous nature of the UART protocol, a certain amount of oversampling must occur on the Receive side of the interface in order to capture the first transition of the ‘Start’ bit. Most hardware IP implementations are configurable between a 13x or a 16x oversample value. Due to the cycle time of the PRU-ICSS (200MHz in most cases) an oversample factor of 8 was chosen for this firmware UART implementation. This 8x oversampling, along with the maximum baud rate desired of 115200, lead to a minimum cycle time for this firmware of 217 PRU cycles:

* 115200 baud \* 8x oversample = 1085.0694 nanoseconds
* 217 PRU cycles \* 200MHz clock rate = 1085 nanoseconds

This means that the firmware needs to have a ‘tick’ that occurs once every 1085 nanoseconds (217 PRU cycles). In this single tick time, the Transmit and Receive state machines of all 3 UART instances needs to complete.

A rolling 16-bit tick counter is incremented after each tick occurs. This rolling tick counter allows each UART to achieve a different baud rate simultaneously. For example, at a baud rate of 115200 each character bit width is equivalent to 8 ticks (hence the 8x oversampling for the maximum supported baud rate). A state machine trying to achieve a baud rate of 115200 would set its next tick to the current tick + 8, a baud rate of 57600 would use current tick + 16, and so on.

### Top Level Flow

The PRU UART firmware top-level program flow is shown in Figure 2.



Figure 2. PRU UART Firmware Top-Level Program Flow

### Wait for Next Tick

Given the requirement that the firmware should not make use of any resources outside of the PRU core, the ‘tick’ interval needs to be tracked using the PRU cycle counter (as opposed to the IEP timer or some other method). Each PRU in the ICSS has its own cycle count register that once enabled, will count up until saturating at 0xFFFFFFFF. The register can be reset to 0 and counting will continue from 1 on the next cycle.

In this firmware, at the end of the ‘Wait for next tick’ state, the PRU cycle counter is reset to 0 and then enabled. The counter then counts up on each cycle through all 6 of the state machines (3 TX and 3 RX) and then is read once the ‘Wait for next tick’ state is entered again. The elapsed cycles are subtracted from the constant 217 cycles (1085 nanoseconds) that have been requested. The remaining cycles are then burned in a tight loop (using only core registers) to only exit the ‘Wait for next tick’ state at the precise time. Using this method, even if all 6 state machines perform their shortest states, the next tick won’t occur until the precise time needed for the next tick.

### Push Transmit Pin Updates & Grab Receive Pin States

The PRU cores in the ICSS have a unique method of generating and detecting general purpose input and output (GPIO). The internal PRU core registers R30 and R31 are used as the GPIO interface.

Any value written to the PRU core R30 register will appear on the GPO pins on the next cycle (5 ns). Bit 0 of R30 corresponds to PRU pin 0, bit 1 corresponds to PRU pin 1 and so on, up to the number of supported PRU outputs on the device.

PRU core register R31 can also be read in a single cycle (5ns) and the current state of the external input pins will be latched. Bit 0 of R31 corresponds to PRU pin 0, bit 1 corresponds to PRU pin 1 and so on, up to the number of supports PRU inputs on the device.

To keep the UART outputs and inputs precise, temporary registers are used to store the next output state and to capture the current input state. Immediately after each tick (every 620ns), the R31 input data is latched into a temporary register (which the RX state machines will use in that tick) and the R30 output data is sent using the values filled in a temporary register (which the TX state machines filled in the previous tick).

### State Machines

There are three transmit and three receive state machine that operate simultaneously and independently of each other. Each state machine stores its state into PRU core registers which persist throughout the firmware life cycle. At the entrance to each state machine there is a check to determine if the current tick (from the rolling 16-bit tick counter) is the next requested tick for this state machine. If it is, then the state machine progresses normally. If it is not, then the firmware moves on to the next state machine. Before exiting each state, the next tick value for the next state is calculated and stored. Using these values, current tick and next tick, each state machine can perform operations at the exact time needed even when each state machine is operating at a different baud rate.

#### Transmit State Machine

Each state begins with the PRU pulling the ‘state’ from DRAM memory (CFG register, RD/WR pointers, pin configuration, and interrupt configuration, current state, tick size, etc.). This is not shown in each state below in Figure 3, but implied.





Figure 3. UART TX State Machine

#### Receive State Machine

Each state begins with the PRU pulling the ‘state’ from DRAM memory (CFG register, RD/WR pointers, pin configuration, and interrupt configuration, current state, tick size, etc.). This is not shown in each state below in Figure 4, but implied.









Figure 4. UART RX State Machine

### Circular Buffers

There is no way for a PRU firmware to emulate the hardware FIFO that is available in the SOC UART IP. To replace the FIFO for transmit and receive data, two circular buffers have been used for each software UART instance. One buffer for transmit and one buffer for receive. A simple read/write pointer mechanism is used by both sides (the PRU and the host processor controlling the UART) to pass data to each other.

The size of the buffers chosen for this firmware are 256 bytes each, with each character taking 2 bytes of space (in order to support 9 bit characters). So each buffer is capable of holding 128 characters at any time.

In order to keep the implementation as simple as possible, a full or empty flag is not being used. In this firmware, if the read pointer is equal to the write pointer then the buffer is empty. This simplification means that 1 character space in the circular buffer is sacrificed since filling the last available space would make the read and write pointers equivalent. The simplified circular buffers in practice can hold 127 characters at a time.

The read and write pointers being stored are byte offsets into the buffer that they are tracking. Only one byte is being allocated for each pointer which caps the maximum buffer size to the 256 bytes which is what was selected in this firmware.

### Binary Configuration Parsing and State Selection

Throughout this firmware it is necessary to parse a configuration from the UART<N>\_CFG register or to determine which state is in effect. One method to do this is to use an assembly instruction to check for equivalence and then branch to an instruction for that specific equivalence. This works well for the first few values being checked for, however the last values being checked are penalized by each miss in front of it. For instance, in the below assembly instructions, if the val\_to\_check register is equal to 1 then the code will branch to label1 after only one instruction. However, if the val\_to\_check register is equal to 4 then it will take 4 instructions to branch to label4.

QBEQ label0, val\_to\_check, 0

QBEQ label1, val\_to\_check, 1

QBEQ label2, val\_to\_check, 2

QBEQ label3, val\_to\_check, 3

This leads to an uneven distribution of cycles based on which value is being checked at the time. It can also make the worst case cycle count grow when the value is set to 4.

Another way to parse the configuration or select the state is to use a binary approach. In this method you can check bits from right to left (bit0, then bit1, and so on) and traverse down a binary tree until you reach your destination. For instance, from the example above, the values 0-3 can be represented by 2 bits. The assembly instructions below can guarantee that each label is reached in exactly 2 cycles every single time:

QBBS \_x1, val\_to\_check, 0

QBBS \_10, val\_to\_check, 1

; Once here the value is 0 since bit 0 nor bit 1 were set

JMP end

\_x1:

QBBS \_11, val\_to\_check, 1

; Once here the value is 1 since bit 0 was set but bit 1 was not set

JMP end

\_10:

; Once here the value is 2 since bit 0 was not set and bit 1 was set

JMP end

\_11:

; Once here the value is 3 since bit 0 and bit 1 were set

JMP end

end:

This method of parsing is used throughout the firmware for parsing the configuration register as well as determining which state is currently active. Without this method, the worst case cycle times for some of the states would have exceeded the maximum cycle budgets allowable by the baud rate and the oversample rate.

## PRU Resource Usage

### PRU Registers

The PRU core has 32 registers. These registers are 1, 2 and 4 byte addressable. The PRU UART firmware uses each register for storing different firmware related information. Table 10 and Table 11 show how the PRU registers are used by the firmware.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Register** | **b3** | **b2** | **b1** | **b0** |
| **R0** | TEMP\_REG | | | |
| **R1** | TEMP2\_REG | | | |
| **R2** | ZERO\_REG | | | |
| **R3** | Tx0:STAT | Tx0:RSVD | Tx0:CONFIG | |
| **R4** | Tx0:PARITY | Tx0:NUM\_BITS | Tx0:CURR\_BIT | Tx0:STATE\_REG |
| **R5** | Tx0:NEXT\_TICK | | Tx0:CHAR\_REG | |
| **R6** | CYCLE\_COUNT\_REG | | Tx0:TICK\_SIZE | |
| **R7** | Tx1:STAT | Tx1:RSVD | Tx1:CONFIG | |
| **R8** | Tx1:PARITY | Tx1:NUM\_BITS | Tx1:CURR\_BIT | Tx1:STATE\_REG |
| **R9** | Tx1:NEXT\_TICK | | Tx1:CHAR\_REG | |
| **R10** | WORST\_DELAY\_REG | | Tx1:TICK\_SIZE | |
| **R11** | Tx2:STAT | Tx2:RSVD | Tx2:CONFIG | |
| **R12** | Tx2:PARITY | Tx2:NUM\_BITS | Tx2:CURR\_BIT | Tx2:STATE\_REG |
| **R13** | Tx2:NEXT\_TICK | | Tx2:CHAR\_REG | |
| **R14** | TICK\_COUNT\_REG | | Tx2:TICK\_SIZE | |
| **R15** | Rx0:RSVD | Rx0:FC\_THRESH | Rx0:CONFIG | |
| **R16** | Rx0:PARITY | Rx0:NUM\_BITS | Rx0:CURR\_BIT | Rx0:STATE\_REG |
| **R17** | Rx0:NEXT\_TICK | | Rx0:CHAR\_REG | |
| **R18** | Rx0:LAST\_TICK | | Rx0:TICK\_SIZE | |
| **R19** | Rx1:RSVD | Rx1:FC\_THRESH | Rx1:CONFIG | |
| **R20** | Rx1:PARITY | Rx1:NUM\_BITS | Rx1:CURR\_BIT | Rx1:STATE\_REG |
| **R21** | Rx1:NEXT\_TICK | | Rx1:CHAR\_REG | |
| **R22** | Rx1:LAST\_TICK | | Rx1:TICK\_SIZE | |
| **R23** | Rx2:RSVD | Rx2:FC\_THRESH | Rx2:CONFIG | |
| **R24** | Rx2:PARITY | Rx2:NUM\_BITS | Rx2:CURR\_BIT | Rx2:STATE\_REG |
| **R25** | Rx2:NEXT\_TICK | | Rx2:CHAR\_REG | |
| **R26** | Rx2:LAST\_TICK | | Rx2:TICK\_SIZE | |
| **R27** | TEMP\_R30 | | | |
| **R28** | TEMP\_R31 | | | |
| **R29** | TEMP CTS/RTS PIN | TEMP Tx/Rx PIN | TEMP Tx/Rx INT Ctrl | TEMP Tx/Rx INT Trigger |
| **R30** | RW: PRU GPO | | | |
| **R31** | R:INTC status & PRU GPI, W: INTC System Event Generation | | | |

Table 15. PRU UART Firmware Register Map

|  |  |  |  |
| --- | --- | --- | --- |
| **FW Register** | **HW Register** | **Purpose** | **Values** |
| TEMP\_REG | R1 | Temporary register |  |
| TEMP2\_REG | R1 | Temporary register |  |
| ZERO\_REG | R2 | Contains 32-bit 0 value. Used for:   * Address generation * Used for writing 0 to 32-bit register | 0x00000000 |
| Tx0:CONFIG | R3.w0 | UART0 Tx:  Copy of 16 LSBs of UART0\_CFG. | Control register, all but bit 11 are used. |
| Tx0:RSVD | R3.b2 | UART0 Tx:  Reserved |  |
| Tx0:STAT | R3.b3 | UART0 Tx:  Internal Status | STAT[0]:TxIntToHostEn  STAT[1]:TxIntToHostSent |
| Tx0:STATE\_REG | R4.b0 | UART0 Tx:  Current state in Tx State Machine. | >=0 and <= 6 |
| Tx0:CURR\_BIT | R4.b1 | UART0 Tx:  Tx bit counter. | >= 0 and <= 9 |
| Tx0:NUM\_BITS | R4.b2 | UART0 Tx:  Number of bits per character. Parsed Tx0:CONFIG. | >= 5 and <= 9 |
| Tx0:PARITY | R4.b3 | UART0 Tx:  Count of 1s in transmitted byte. Used for Tx parity calculation. | >= 0 and <= 9 |
| Tx0:CHAR\_REG | R5.w0 | UART0 Tx:  Current byte for Tx. | xxxxxxx000000000b  to  xxxxxxx111111111b |
| Tx0:NEXT\_TICK | R5.w2 | UART0 Tx:  Next system tick to execute Tx State Machine. | Any 16-bit value |
| CYCLE\_COUNT\_REG | R6.w2 | Number of PRU clock cycles corresponding to a system tick. | 124 |
| Tx0:TICK\_SIZE | R6.w0 | UART0 Tx:  Number of system ticks for configured baud rate. | Min: 14 (115200)  Max: 5376 (300) |
| Tx1:CONFIG | R7.w0 | UART1 Tx:  Copy of 16 LSBs of UART1\_CFG | Control register, all but bit 11 are used. |
| Tx1:RSVD | R7.b2 | UART1 Tx:  Reserved |  |
| Tx1:STAT | R7.b3 | UART1 Tx:  Internal Status | STAT[0]:TxIntToHostEn  STAT[1]:TxIntToHostSent |
| Tx1:STATE\_REG | R8.b0 | UART1 Tx:  Current state in Tx State Machine. | >=0 and <= 6 |
| Tx1:CURR\_BIT | R8.b1 | UART1 Tx:  Tx bit counter. | >= 0 and <= 9 |
| Tx1:NUM\_BITS | R8.b2 | UART1 Tx:  Number of bits per character. Parsed Tx1:CONFIG. | >= 5 and <= 9 |
| Tx1:PARITY | R8.b3 | UART1 Tx:  Count of 1s in transmitted byte. Used for Tx parity calculation. | >= 0 and <= 9 |
| Tx1:CHAR\_REG | R9.w0 | UART1 Tx:  Current byte for Tx. | xxxxxxx000000000b  to  xxxxxxx111111111b |
| Tx1:NEXT\_TICK | R9.w2 | UART1 Tx:  Next system tick to execute Tx State Machine. | Any 16-bit value |
| Tx1:TICK\_SIZE | R10.w0 | UART1 Tx:  Number of system ticks for configured baud rate. | Min: 14 (115200)  Max: 5376 (300) |
| WORST\_DELAY\_REG | R10.w2 | wait\_tick: used to determine if loop iteration was over cycle budget. | 0x0FFF |
| Tx2:CONFIG | R11.w0 | Copy of 16 LSBs of UART2\_CFG for Tx | Control register, all but bit 11 are used. |
| Tx2:RSVD | R11.b2 | UART2 Tx:  Reserved |  |
| Tx2:STAT | R11.b3 | UART2 Tx:  Internal Status | STAT[0]:TxIntToHostEn  STAT[1]:TxIntToHostSent |
| Tx2:STATE\_REG | R12.b0 | UART2 Tx:  Current state in Tx State Machine. | >=0 and <= 6 |
| Tx2:CURR\_BIT | R12.b1 | UART2 Tx:  Tx bit counter. | >= 0 and <= 9 |
| Tx2:NUM\_BITS | R12.b2 | UART2 Tx:  Number of bits per character. Parsed Tx2:CONFIG. | >= 5 and <= 9 |
| Tx2:PARITY | R12.b3 | UART2 Tx:  Count of 1s in transmitted byte. Used for Tx parity calculation. | >= 0 and <= 9 |
| Tx2:CHAR\_REG | R13.w0 | UART2 Tx:  Current byte for Tx. | xxxxxxx000000000b  to  xxxxxxx111111111b |
| Tx2:NEXT\_TICK | R13.w2 | UART2 Tx:  Next system tick to execute Tx State Machine. | Any 16-bit value |
| Tx2:TICK\_SIZE | R14.w0 | UART2 Tx:  Number of system ticks for configured baud rate. | Min: 14 (115200)  Max: 5376 (300) |
| TICK\_COUNT\_REG | R14.w2 | System tick counter. | 0 to 0xFFFF |
| Rx0:CONFIG | R15.w0 | Copy of 16 LSBs of UART0\_CFG for Rx | Control register, all but bit 11 are used. |
| Rx0:FC\_THRESH | R15.b2 | UART0 Rx:  Rx hardware flow control threshold. | > 0 and <= 255 |
| Rx0:RSVD | R15.b3 | Reserved |  |
| Rx0:STATE\_REG | R16.b0 | UART0 Rx:  Current state in Rx State Machine. | >= 0 and <= 8 |
| Rx0:CURR\_BIT | R16.b1 | UART0 Rx:  Rx bit counter. | >= 0 and <= 9 |
| Rx0:NUM\_BITS | R16.b2 | UART0 Rx:  Number of bits per character. Parsed Rx0:CONFIG. | >= 5 and <= 9 |
| Rx0:PARITY | R16.b3 | UART0 Rx:  Count of 1s in received byte. Used for Rx parity calculation. | >= 0 and <= 9 |
| Rx0:CHAR\_REG | R17.w0 | UART0 Rx:  Current byte for Rx. | xxxxxxx000000000b  to  xxxxxxx111111111b |
| Rx0:NEXT\_TICK | R17.w2 | UART0 Rx:  Next system tick to execute Rx State Machine. | Any 16-bit value |
| Rx0:TICK\_SIZE | R18.w0 | UART0 Rx:  Number of system ticks for configured baud rate. | Min: 14 (115200)  Max: 5376 (300) |
| Rx0:LAST\_TICK | R18.w2 | UART0 Rx:  Last system tick for Stop bit. | Any 16-bit value |
| Rx1:CONFIG | R19.w0 | Copy of 16 LSBs of UART1\_CFG for Rx | Control register, all but bit 11 are used. |
| Rx1:FC\_THRESH | R19.b2 | UART1 Rx:  Rx hardware flow control threshold. | > 0 and <= 255 |
| Rx1:RSVD | R19.b3 | Reserved |  |
| Rx1:STATE\_REG | R20.b0 | UART1 Rx:  Current state in Rx State Machine. | >= 0 and <= 8 |
| Rx1:CURR\_BIT | R20.b1 | UART1 Rx:  Rx bit counter. | >= 0 and <= 9 |
| Rx1:NUM\_BITS | R20.b2 | UART1 Rx:  Number of bits per character. Parsed Rx1:CONFIG. | >= 5 and <= 9 |
| Rx1:PARITY | R20.b3 | UART1 Rx:  Count of 1s in received byte. Used for Rx parity calculation. | >= 0 and <= 9 |
| Rx1:CHAR\_REG | R21.w0 | UART1 Rx:  Current byte for Rx. | xxxxxxx000000000b  to  xxxxxxx111111111b |
| Rx1:NEXT\_TICK | R21.w2 | UART1 Rx:  Next system tick to execute Rx State Machine. | Any 16-bit value |
| Rx1:TICK\_SIZE | R22.w0 | UART1 Rx:  Number of system ticks for configured baud rate. | Min: 14 (115200)  Max: 5376 (300) |
| Rx1:LAST\_TICK | R22.w2 | UART1 Rx:  Last system tick for Stop bit. | Any 16-bit value |
| Rx2:CONFIG | R23.w0 | Copy of 16 LSBs of UART2\_CFG for Rx | Control register, all but bit 11 are used. |
| Rx2:FC\_THRESH | R23.b2 | UART2 Rx:  Rx hardware flow control threshold. | > 0 and <= 255 |
| Rx2:RSVD | R23.b3 | Reserved |  |
| Rx2:STATE\_REG | R24.b0 | UART2 Rx:  Current state in Rx State Machine. | >= 0 and <= 8 |
| Rx2:CURR\_BIT | R24.b1 | UART2 Rx:  Rx bit counter. | >= 0 and <= 9 |
| Rx2:NUM\_BITS | R24.b2 | UART2 Rx:  Number of bits per character. Parsed Rx2:CONFIG. | >= 5 and <= 9 |
| Rx2:PARITY | R24.b3 | UART2 Rx:  Count of 1s in received byte. Used for Rx parity calculation. | >= 0 and <= 9 |
| Rx2:CHAR\_REG | R25.w0 | UART2 Rx:  Current byte for Rx. | xxxxxxx000000000b  to  xxxxxxx111111111b |
| Rx2:NEXT\_TICK | R25.w2 | UART2 Rx:  Next system tick to execute Rx State Machine. | Any 16-bit value |
| Rx2:TICK\_SIZE | R26.w0 | UART2 Rx:  Number of system ticks for configured baud rate. | Min: 14 (115200)  Max: 5376 (300) |
| Rx2:LAST\_TICK | R26.w2 | UART2 Rx:  Last system tick for Stop bit. | Any 16-bit value |
| TEMP\_R30 | R27 | Temporary R30 register. Holds PRU GPO outputs for current loop iteration. |  |
| TEMP\_R31 | R28 | Temporary R31 register. Holds PRU GPI inputs for current loop iteration. |  |
| TEMP\_PIN\_INT\_CFG | R29 | Temporary PIN & INT configuration register for TX/RX |  |

Table 16. PRU UART Firmware Register Usage

### PRU Memory

Instruction Memory (IMEM)

The UART PRU firmware IMEM requirements are presented below in Table 12.

|  |  |
| --- | --- |
| **No. UART Instances** | **Required IMEM (bytes)** |
| 1 | 1828 |
| 2 | 3276 |
| 3 | 4724 |

Table 17. PRU UART Firmware IMEM Requirements

Data Memory (DMEM)

The UART PRU firmware requires:

* 32 bytes of DMEM for global configuration/control/status information.
* 5\*4 + 256 + 256 bytes = 532 bytes of DMEM per UART instance.

The UART PRU firmware DMEM requirements are presented below in Table 13.

|  |  |
| --- | --- |
| **No. UART Instances** | **Required IMEM (bytes)** |
| 1 | 564 |
| 2 | 1096 |
| 3 | 1624 |

Table 18. PRU UART Firmware DMEM Requirements

### PRU Cycles

#### Worst-Case Cycle Counts

The worst-case PRU UART FW cycle counts can be expressed using the equation:

*Cwc* = *Cml* + *Ntx*\* max(*Ctx*) + *Nrx* \* max(*Crx*)

Table 14 provides descriptions of the terms in this equation, along with values for the terms. Details concerning the UART TX and RX state machine cycle counts are discussed in following sections.

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **Value** |
| *Cml* | Main loop cycles outside of UART TX/RX macro calls | 11 |
| *Ntx* | Number of UART TX instances | 3 |
| *max(Ctx)* | Maximum total state cycles over all TX states | 21 |
| *Nrx* | Number of UART RX instances | 3 |
| *max(Crx)* | Maximum total state cycles over all RX states | 22 |
| *Cwc* | Worst-case cycle counts | 140 |

Table 19. Firmware Cycle Counts Equation Terms

#### Transmit Cycle Counts

Transmit Overhead Cycles

All UART transmit states include 6 overhead cycles:

*Ctxovr* = 1 + 3 + 1 + 1 = 6 cycles

Overhead cycles include:

* Check if execution tick: 1 cycle
* Decode/enter state: 3 cycles
* Update next tick: 1 cycle
* Exit state: 1 cycle

Worst-Case Transmit State Cycles

Table 15 presents worst-case UART transmit state cycles, and worst-case total UART transmit cycles, where the worst-case total state cycles are the sum of the transmit and transmit overhead state cycles.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **State** | **Description** | **Worst‑Case  Cycles (*Ctxstate*)** | **Overhead**  **Cycles (*Ctxovr*)** | **Total Worst‑Case  Cycles**  **(*Ctx*)** |
| 0 | Parse UART configuration | 15 | 6 | 21 |
| 1 | Check if character available for Tx | 11 | 6 | 17 |
| 2 | Tx Start bit | 11 | 6 | 17 |
| 3 | Tx character data bit | 9 | 6 | 15 |
| 4 | Calculate & Tx Parity bit | 8 | 6 | 14 |
| 5 | Tx Stop bit | 10 | 6 | 16 |
| 6 | Determine Stop bit duration | 6 | 6 | 12 |
| 7 | Reset | 16 | 6 | 22 |

Table 20. Worst-Case UART Transmit State Cycles

Excluding the Reset state cycles (the Reset state is not executed during normal transmit operations), it can be observed that max(*Ctx*) = 21. This maximum occurs for STATE0 when the Host removes the UART from Reset.

#### Receive Cycle Counts

Receive Overhead Cycles

All UART receive states include 7 overhead cycles:

*Crxovr* = 1 + 4 + 1 + 1 = 7 cycles

Overhead cycles include:

* Check if execution tick: 1 cycle
* Decode/enter state: 3 cycles
* Update next tick: 1 cycle
* Exit state: 1 cycle

Worst-Case Receive State Cycles

Table 16 present worst-case UART receive state cycles, and worst-case total UART receive cycles, where the worst-case total state cycles are the sum of the receive and receive overhead state cycles.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **State** | **Description** | **Worst-Case Cycles (*Crxstate*)** | **Overhead Cycles  (*Crxovr*)** | **Total Worst‑Case Cycles**  **(*Crx*)** |
| 0 | Wait for Rx pin high | 6 | 7 | 13 |
| 1 | Check for Start bit | 15 | 7 | 22 |
| 2 | Check if space in Rx Buffer for incoming character | 10 | 7 | 17 |
| 3 | Check for FALSE Start bit | 10 | 7 | 17 |
| 4 | Rx character data bit | 9 | 7 | 16 |
| 5 | Rx Parity bit & determine if Parity error | 9 | 7 | 16 |
| 6 | Configure LAST\_TICK based on Stop bit duration | 8 | 7 | 15 |
| 7 | Check for Framing error | 3 | 7 | 10 |
| 8 | Write Rx character to Rx Buffer, update Rx Buffer write index | 15 | 6\* | 21 |
| 15 | Reset | 17 | 7 | 24 |

\*Note: Rx State8 exits without an explicit jump, so a single cycle is removed from the overhead cycles for this State.

Table 21. Worst-Case UART Receive State Cycles

Excluding the Reset state cycles (the Reset state is not executed during normal receive operations), it can be observed that max(*Crx*) = 22. This maximum occurs for STATE1 when the Host removes the UART from Reset.

#### Wait Tick Cycle Counts

The wait\_tick macro waits for unused cycles in each main loop iteration. This macro computes the difference between the system tick cycle count and the measured number of main loop cycles for a particular main loop iteration, i.e. (217 – measured) cycles, and waits (loops) until this number of cycles has expired. Hence the main loop always consumes the system tick cycle count for each loop iteration.

Wait\_tick and total main loop cycles can be expressed as:

*Cwt* = 13 + (*Crem* – 12) = 13 + (*Cst* – CYCLE – 12) = 218 – CYCLE

*Ctot*= *Cml* + *Cwt* = CYCLE + (218 – CYCLE) = 218

|  |  |
| --- | --- |
| **Term** | **Description** |
| CYCLE | Measured cycle count for main loop iteration |
| *Cst* | System tick cycles |
| *Crem* | Difference between 217 and CYCLE for loop iteration |
| *Cwt* | wait\_tick cycles for loop iteration |
| *Cml* | Cycles for loop iteration ignoring wait\_tick |
| *Ctot* | Total cycles for loop iteration |

## Firmware Source Code

### Firmware Macros Description

Table 17 lists the macros used in the firmware source code.

|  |  |  |
| --- | --- | --- |
| **Macros** | **File** | **Function** |
| **config\_bit** | PRU\_SW\_UART.asm | Poke bit into TEMP\_R30 (temp output register) position |
| **read\_bit** | PRU\_SW\_UART.asm | Pick bit from TEMP\_R31 (temp input register) position |
| **wait\_tick** | PRU\_SW\_UART.asm | Wait for (217 – measured) cycles to achieve 217 cycles total loop iteration time. |
| **parse\_baud\_rate** | PRU\_SW\_UART.asm | Parse baud rate configuration |
| **parse\_num\_bits** | PRU\_SW\_UART.asm | Parse number of bits configuration |
| **tx\_macro** | PRU\_SW\_UART.asm | UART instance transmit |
| **rx\_macro** | PRU\_SW\_UART.asm | UART instance receive |

Table 22. Firmware Macros

### Firmware Sources Description

Table 18 lists the firmware source code files.

|  |  |
| --- | --- |
| **File** | **Description** |
| **AM335x\_PRU.cmd** | Linker command file for firmware build |
| **icss\_ctrl\_regs.h** | Header file, contains ICSS PRU control register definitions |
| **icss\_defines.h** | Header file, contains ICSS global definitions |
| **icss\_uart.h** | Header file, contains UART firmware register definitions |
| **PRU\_SW\_UART.asm** | Assembly source code for TX/RX UART SW IP instances |

Table 23. Firmware Source Files

# RTOS Driver Support

## External APIs

The UART Low-Level Driver (LLD) supports an API for UART hardware IPs. An application uses this API (calls API functions) to configure and use UART hardware IPs. The UART LLD supports the same API for PRU UART firmware software IPs, and an application uses the API in the same manner for using UART software and hardware IPs. The tables below contains a list of the API functions.

|  |  |
| --- | --- |
| **API Function** | **Functional Description** |
| **UART\_close** | Closes UART peripheral specified by UART handle |
| **UART\_control** | Performs implementation-specific control for UART specified by UART\_Handle |
| **UART\_init** | Initializes UART module |
| **UART\_open** | Initializes UART peripheral specified by index value. Parameter specify UART operational mode. |
| **UART\_params\_init** | Initializes UART\_Params structure to its defaults |
| **UART\_write** | Initiate write operation (BLOCKING, CALLBACK, or POLLING mode) to UART controller specified by UART\_Handle |
| **UART\_writePolling** | Initiate write operation in POLLING mode for UART controller specified by UART\_Handle |
| **UART\_writeCancel** | Cancel write operation in CALLBACK mode for UART controller specified by UART\_Handle |
| **UART\_read** | Initiate read operation (BLOCKING, CALLBACK, or POLLING mode) to UART controller specified by UART\_Handle |
|  | Initiate read operation in POLLING mode for UART controller specified by UART\_Handle |
| **UART\_readCancel** | Cancel read operation in CALLBACK mode for UART controller specified by UART\_Handle |
| **UART\_read2** | Initiate read operation (BLOCKING, CALLBACK, or POLLING mode) to UART controller specified by UART\_Handle |
| **UART\_write2** | Initiate write operation (BLOCKING, CALLBACK, or POLLING mode) to UART controller specified by UART\_Handle |
| **UART\_transactionInit** | Initialize UART\_Transaction to its defaults |

Table 24. UART LLD API Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Return Type** | **API Function** | **Arguments** | **Functional Description** |
| void | **UART\_close** | handle | A UART\_Handle returned from UART\_open() |
| int32\_t | **UART\_control** | handle | A UART\_Handle returned from UART\_open() |
| uint32\_t | A command value defined by the driver specific implementation |
| void \* | An optional R/W (read/write) argument that accompanies cmd |
| void | **UART\_init** | void | The UART\_config structure must exist and be persistent before this function can be called |
| UART\_Handle | **UART\_open** | uint32\_t | Logical peripheral number for the UART indexed into the UART\_config table |
| UART\_Params \* | Pointer to an parameter block, if NULL it will use default values. All the fields in this structure are RO (read-only). |
| void | **UART\_Params\_init** | UART\_Params\* | An pointer to UART\_Params structure for initialization |
| int32\_t | **UART\_write** | handle | A UART\_Handle returned from UART\_open() |
| void \* | Pointer to buffer containing data to be written |
| size\_t | Number of bytes to write to UART |
| int32\_t | **UART\_writePolling** | handle | A UART\_Handle returned from UART\_open() |
| void \* | Pointer to buffer containing data to be written |
| size\_t | Number of bytes to write to UART |
| void | **UART\_writeCancel** | handle | A UART\_Handle returned from UART\_open() |
| int32\_t | **UART\_read** | handle | A UART\_Handle returned from UART\_open() |
| void \* | Pointer to buffer to which data will be written |
| size\_t | Number of bytes to read from UART |
| int32\_t | **UART\_readPolling** | handle | A UART\_Handle returned from UART\_open() |
| void \* | Pointer to buffer to which data will be written |
| size\_t | Number of bytes to read from UART |
| void | **UART\_readCancel** | handle | A UART\_Handle returned from UART\_open() |
| int32\_t | **UART\_read2** | handle | A UART\_Handle returned from UART\_open() |
| UART\_Transaction \* | Pointer to UART\_Transaction containing parameters for read. All fields within transaction are write-only unless noted in the driver implementation. |
| int32\_t | **UART\_write2** | handle | A UART\_Handle returned from UART\_open() |
| UART\_Transaction \* | Pointer to UART\_Transaction containing parameters for write. All fields within transaction are write-only unless noted in the driver implementation. |
| void | **UART\_transactionInit** | UART\_Transaction \* | Pointer to UART\_Transaction to initialize |

Table 25. UART LLD API Function Details

## Internal Files & Functions

The UART LLD API is supported on different versions of UART hardware IP. The LLD API internal implementation is different for different UART hardware IPs, and the internal file and function organization is also different. This is also the case for PRU UART firmware software IPs. The LLD API for UART software IP is implemented in the source file UART\_v2.c. There many LLD internal functions which are driver implementation dependent. The table below shows the relationship between the API functions and these internal driver functions.

|  |  |
| --- | --- |
| **External API Function** | **Mapped Internal Implementation Function** |
| UART\_close | UART\_close\_v2 |
| UART\_control | UART\_control\_v2 |
| UART\_init | UART\_init\_v2 |
| UART\_open | UART\_open\_v2 |
| UART\_read | UART\_read\_v2 |
| UART\_readPolling | UART\_readPolling\_v2 |
| UART\_readCancel | UART\_readCancel\_v2 |
| UART\_write | UART\_write\_v2 |
| UART\_writePolling | UART\_writePolling\_v2 |
| UART\_writeCancel | UART\_writeCancel\_v2 |
| UART\_read2 | UART\_read2\_v2 |
| UART\_write2 | UART\_write2\_v2 |
| - | UART\_v2\_initIcssPru |
| - | UART\_v2\_deinitIcssPru |
| - | UART\_v2\_resetObject |
| - | UART\_v2\_initFwGlobalCfg |
| - | UART\_v2\_waitFwInitNotify |
| - | UART\_v2\_enablePru |
| - | UART\_v2\_disablePru |
| - | UART\_v2\_resetPru |
| - | UART\_v2\_disableInst |
| - | UART\_v2\_checkInstDisabled |
| - | UART\_v2\_enableInst |
| - | UART\_v2\_waitInstEnableAck |
| - | UART\_v2\_disableInstTxInt |
| - | UART\_v2\_enableInstTxInt |
| - | UART\_v2\_disableInstRxInt |
| - | UART\_v2\_enableInstRxInt |
| - | UART\_v2\_configInstPinMux |
| - | UART\_v2\_configInst |
| - | UART\_v2\_configInstPins |
| - | UART\_v2\_configInstSysEvts |
| - | UART\_v2\_clearInstTxBuffer |
| - | UART\_v2\_clearInstRxBuffer |
| - | UART\_v2\_pruGPIOMuxConfig |
| - | UART\_v2\_configInstBaudRate |
| - | xlateUartBaudRateToFwBaudRate |
| - | UART\_v2\_configInstNumBitsPerChar |
| - | xlateUartDatalenToFwNumBitsPerChar |
| - | UART\_v2\_configInstStopBitSize |
| - | xlateUartStopBitsToFwStopBitSize |
| - | UART\_v2\_configInstParity |
| - | UART\_v2\_configInstHwFlowControl |
| - | UART\_v2\_cancelInstWriteNoCallback |
| - | UART\_v2\_cancelInstReadNoCallback |
| - | UART\_v2\_putInstCharTxBuffTimeout |
| - | UART\_v2\_getInstCharRxBuffTimeout |
| - | UART\_v2\_putInstCharTxBuff |
| - | UART\_v2\_putInstCharTxBuffNonBlock |
| - | UART\_v2\_putInstCharTxBuffBlock |
| - | UART\_v2\_getInstTxBuffFullStatus |
| - | UART\_v2\_getInstTxBuffEmptyStatus |
| - | UART\_v2\_getInstCharRxBuffNonBlock |
| - | UART\_v2\_callback |
| - | UART\_v2\_hwiIntFxn |

Table 26. UART LLD Software IP Internal Functions

# Test Plan

## EVM Support

### bbbAM335x

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ICSS** | **PRU** | **Instance** | **Functional Pin** | **PRU GPIO Pins** | **EVM Port** | **EVM pin** |
| ICSS1 | PRU0 | UART0 | TX | pr1\_pru0\_pru\_r30\_0 | P9 | 31 |
| RX | pr1\_pru0\_pru\_r31\_1 | P9 | 29 |
| CTS | - | - | - |
| RTS | - | - | - |
| ICSS1 | PRU0 | UART1 | TX | pr1\_pru0\_pru\_r30\_2 | P9 | 30 |
| RX | pr1\_pru0\_pru\_r31\_3 | P9 | 28 |
| CTS | - | - | - |
| RTS | - | - | - |
| ICSS1 | PRU0 | UART2 | TX | pr1\_pru0\_pru\_r30\_5 | P9 | 27 |
| RX | pr1\_pru0\_pru\_r31\_7 | P9 | 25 |
| CTS | - | - | - |
| RTS | - | - | - |
| ICSS1 | PRU1 | UART0 | TX | pr1\_pru1\_pru\_r30\_0 | P8 | 45 |
| RX | pr1\_pru1\_pru\_r31\_1 | P8 | 46 |
| CTS | pr1\_pru1\_pru\_r31\_2 | P8 | 43 |
| RTS | pr1\_pru1\_pru\_r30\_3 | P8 | 44 |
| ICSS1 | PRU1 | UART1 | TX | pr1\_pru1\_pru\_r30\_4 | P8 | 41 |
| RX | pr1\_pru1\_pru\_r31\_5 | P8 | 42 |
| CTS | pr1\_pru1\_pru\_r31\_6 | P8 | 39 |
| RTS | pr1\_pru1\_pru\_r30\_7 | P8 | 40 |
| ICSS1 | PRU1 | UART2 | TX | pr1\_pru1\_pru\_r30\_8 | P8 | 27 |
| RX | pr1\_pru1\_pru\_r31\_9 | P8 | 29 |
| CTS | pr1\_pru1\_pru\_r31\_10 | P8 | 28 |
| RTS | pr1\_pru1\_pru\_r30\_11 | P8 | 30 |

Table 27. bbbAM335x UART Instances

## Test Setup

### bbbAM335x

The PRU UART firmware Unit Test setup requires the following hardware:

* AM335x BeagleBone Black (BBB), see <https://beagleboard.org/black>
* Standard FTDI cable
  + See <https://elinux.org/Beagleboard:BeagleBone_Black_Accessories>
  + Pin 1 on the cable is the black wire
* PC with USB

The Unit Test requires these connections:

* PRU0 UART0 to AM335x UART1 HW IP in external loopback (SW IP to HW IP)
* PRU0 UART1 to PRU0 UART2 in external loopback (SW IP to SW IP)
* PRU1 UART1 to PRU1 UART2 in external loopback (SW IP to SW IP)
* PRU1 UART0 to FTDI cable, FTDI cable to PC USB (SW IP to PC UART)

The required connections are shown in Figure 5.



Figure 5. PRU Firmware Unit Test Setup

## Unit Test

The Unit Test tests all features of the PRU UART firmware. It tests all UART instances and settings.

### Unit Test Build

The Unit Test is built using the following steps:

1. Open a DOS shell
2. Change directory to PDK packages directory:  
   > cd <PDK install directory>\packages
3. Execute PDK environment creation script:  
   > pdksetupenv.bat
4. Build UART-LLD & PRU UART firmware:  
   > gmake LIMIT\_SOCS=”am335x” uart
5. Generate the CCS project for the Unit Test  
   > pdkProjectCreate.bat AM335x bbbAM335x little uart test arm
6. Execute CCS, import Unit Test project
   * Project→Import CCS Projects
   * Click “Browse” button next to “Select search directory”
   * Browse to <PDK install directory>\packages\MyExampleProjects
   * Select “UART\_FwTest\_bbbAM335x\_TestProject”
   * Click “Finish” button
7. Build CCS project for Unit Test
   * Right-click on project in Project Explorer window
   * Click “Rebuild Project” from context sensitive window
   * The Unit Test executable “UART\_FwTest\_bbbAM335x\_armExampleProject.out” is located in folder <PDK install directory>\packages\MyExampleProjects\UART\_FwTest\_bbbAM335x\_armTestProject\Debug

Note the build steps described above are for Windows. However, the Unit Test can be built on Linux using a similar procedure.

### Unit Test Log

The Unit Test prints a log to the UART connected to the FTDI cable (see Figure 5). Output is written to the log for each executed test. An example of the log is shown below.

UART UT 1

UART read write test in block mode

enter the data of 16 character

Data entered is as follows

0123456789012345

enter the data of 16 character

Data entered is as follows

9876543210987654

UART UT 1 PASSED

UART UT 2

UART read write test in callback mode

enter the data of 16 character

Data entered is as follows

0123456789012345

enter the data of 16 character

Data entered is as follows

9876543210987654

UART UT 2 PASSED

UART UT 3

UART timeout test, wait for 10 seconds to timeout read

Read timed out

UART UT 3 PASSED

UART UT 4

UART RX error test, enter a break

UART UT 4 PASSED

UART UT 5

UART read write cancel test, enter less than 16 chars

Previous read canceled

enter the data of 16 character

Data entered is as follows

0123456789012345

Previous write canceled

Previous write canceled

UART UT 5 PASSED

UART UT 6

UART simultaneous read write test

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

enter the data of 16 character

Data entered is as follows

9876543210987654

UART UT 6 PASSED

UART UT 7

UART stdio printf and scanf test

enter the data of 16 character and press ENTER

0123456789012345

UART UT 7 PASSED

UART UT 8

UART polling timeout test, wait for 10 seconds to timeout read

Read timed out

abcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyzabcdwxyza

Read timed out

enter the data of 16 character

Data entered is as follows

0123456789012345

UART UT 8 PASSED

UART UT 9

UART stdio printf and scanf test with STDIO params(Default)

enter the data of 16 character and press ENTER

9876543210987654

UART UT 9 PASSED

UART UT 10

UART read write test with interrupt disabled

enter the data of 16 character

Data entered is as follows

0123456789012345

enter the data of 16 character

Data entered is as follows

9876543210987654

UART UT 10 PASSED

UART UT 11

UART HW IP write to SW IP read test, blocking, external loopback

UART UT 11 PASSED

UART UT 12

UART SW IP write to HW IP read test, blocking, external loopback

UART UT 12 PASSED

UART UT 13

UART HW IP write to SW IP read test, callback, external loopback

UART UT 13 PASSED

UART UT 14

UART SW IP write to HW IP read test, callback, external loopback

UART UT 14 PASSED

UART UT 15

UART SW IP5 write to SW IP6 read test, blocking, external loopback

UART UT 15 PASSED

UART UT 16

UART SW IP3 write to SW IP2 read test, blocking, external loopback

UART UT 16 PASSED

UART UT 17

UART SW IP5 write to SW IP6 read test, callback, external loopback

UART UT 17 PASSED

UART UT 18

UART SW IP3 write to SW IP2 read test, callback, external loopback

UART UT 18 PASSED

UART UT 19

UART SW IP5 write to SW IP6 read test, callback, HW FC, external loopback

UART UT 19 PASSED

UART UT 20

UART open/close SW IP test, external loopback

UART UT 20 PASSED

All tests have passed.